

Informal Concepts in Machines*

Kurt Ammon[†]

Abstract

This paper constructively proves the existence of an effective procedure generating a computable (total) function that is not contained in any given effectively enumerable set of such functions. The proof implies the existence of machines that process informal concepts such as computable (total) functions beyond the limits of any given Turing machine or formal system, that is, these machines can, in a certain sense, “compute” function values beyond these limits. We call these machines creative. We argue that any “intelligent” machine should be capable of processing informal concepts such as computable (total) functions, that is, it should be creative. Finally, we introduce hypotheses on creative machines which were developed on the basis of theoretical investigations and experiments with computer programs. The hypotheses say that machine intelligence is the execution of a self-developing procedure starting from any universal programming language and any input.

1 Introduction

Hilbert’s program aimed to reduce mathematics to a formal system in order to avoid inconsistencies in mathematics. In particular, Hilbert’s Entscheidungsproblem (decision problem) aimed to find “a procedure that allows one to decide on the validity, respectively satisfiability, of a given logical expression by a finite number of operations”.¹ In order to prove that Hilbert’s

*This work is licensed under the Creative Commons Attribution-No Derivative Works 3.0 Unported License (see <http://creativecommons.org/licenses/by-nd/3.0/>).

[†]Correspondence to paper at cstruct.org. Comments are welcome.

¹Hilbert and Ackermann [1928, p. 73]: ein Verfahren ..., das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheidung über die Allgemeingültigkeit bzw. Erfüllbarkeit erlaubt.

Entscheidungsproblem is unsolvable, Turing [1936] introduced his “computing machines” which are a formalization of a *procedure* in Hilbert’s sense. Gödel [1965, p. 72] writes:

Turing’s work gives an analysis of the concept of “mechanical procedure” (alias “algorithm” ...) ... This concept is shown to be equivalent with that of a “Turing machine”. A formal system can simply be defined to be any mechanical procedure for producing formulas, called provable formulas.

Hopcroft and Ullman [1979, p. 147] write that “the Turing machine is equivalent in computing power to the digital computer as we know it today”. They implicitly assume that the computer is *used* for executing a *given procedure* or program that was developed manually. Turing [1986] asks whether a computer can be *used* in another way:

It has been said that computing machines can only carry out the processes that they are instructed to do. ... Up till the present machines have only be been used in this way. But is it necessary that they should always be used in such a manner?

Turing [1969] discusses the development of intelligence in man and in machines:

If the untrained infant’s mind is to become an intelligent one, it must acquire both discipline and initiative. So far we have been considering only discipline. To convert a brain or machine into a universal machine is the extremest form of discipline. But discipline is certainly not enough in itself to produce intelligence. That which is required in addition we call initiative. ... Our task is to discover the nature of this residue as it occurs in man, and to try and copy it in machines.

We investigate “the nature of this residue” called “initiative” (see Sieg, 1994, Section 5, Final remarks). Section 2 proves the existence of an effective procedure generating a computable (total) function that is not contained in any given effective enumeration of such functions. This procedure can be regarded as a bridge to the informal concept of computable (total) functions, that is, to Turing’s uncomputable residue. On the basis of this procedure Section 3 defines creative machines which can, in a certain sense, “compute” function values beyond the limits of any given Turing machine. Section 4

introduces hypotheses on creative machines which say that Turing's uncomputable residue is the execution of a self-developing procedure starting from any universal programming language and any input. This process, which produces formally irreducible experience, can be regarded as a new use of computers. The remaining sections discuss our proof, creative machines and related work.

2 First Theorems

In this and the following sections we simply write *computable function* for an effectively computable *total* function of natural numbers which is defined for *all* natural numbers.

Theorem 1 *There is an effective procedure generating a computable function that is not contained in any given effective enumeration of such functions.*

Proof. Let f_1, f_2, \dots be an effective enumeration of computable functions. We define a new function g by

$$g(n) = f_n(n) + 1 \tag{1}$$

for all natural numbers n . Obviously, $g(n)$ is defined for all natural numbers n because $f_i(n)$ is defined for all natural numbers i and all natural numbers n . Furthermore, g is computable because f_1, f_2, \dots is an effective enumeration of computable functions according to our original assumption. The expression $f_n(n) + 1$ in the definition (1) of g can be regarded as a functional pseudocode, that is, as a computer program, say R , that computes the function g for all natural numbers n . There is an effective procedure that generates the program R . This procedure can be represented as a computer program whose input is the effective enumeration f_1, f_2, \dots , that is, a program, say E , generating the functions f_1, f_2, \dots , and whose output is the program R . In order to generate $g(n)$ from any natural number n , the program R thus generates the function f_n by applying E to n and then adds 1 to the result of applying f_n to n . Because of definition (1), $g(n)$ is different from $f_n(n)$ for all natural numbers n . This implies that the computable function g is different from all functions f_n , where n is any natural number. Therefore, there is an effective procedure generating a computable function g that is not contained in any given effective enumeration of such functions.

Theorem 2 *There is an effective procedure generating a computable function that is not contained in any given formal system with a predicate for such functions.*

Proof. Let S be a formal system with a predicate Q for computable functions. Because a formal system can be regarded as a mechanical procedure or algorithm producing provable formulas (Gödel 1965, p. 72), the formal system S produces an effective enumeration of provable formulas $Q(f_1), Q(f_2), \dots$ that contains an effective enumeration of all computable functions f_1, f_2, \dots in S . According to Theorem 1 there is an effective procedure generating a computable function that not contained in the effective enumeration of all computable functions f_1, f_2, \dots in S .

Theorem 2 implies that the concept of computable functions is informal in the sense that the effective procedure in the theorem generates a computable function beyond the limits of any given formal system.

3 Creative Machines

We apply Theorem 1 to a hypothetical machine C capable of processing an effective procedure that exists according to Theorem 1. There are no assumptions on the internal structure of C . In particular, it is not assumed that C is a Turing machine in any sense.

Theorem 3 *Let C be a machine capable of processing an effective procedure P that exists according to Theorem 1. Then, there is no Turing machine generating all computable functions that C can generate by means of P .²*

Proof. Let C be a machine capable of processing an effective procedure P that exists according to Theorem 1 and let T be any Turing machine generating any enumeration f_1, f_2, \dots of computable functions. The enumeration f_1, f_2, \dots is effective because it is generated by a Turing machine. The machine C can generate a function g by applying the effective procedure P to this effective enumeration. According to Theorem 1 the function g is a computable function that is not contained in the enumeration f_1, f_2, \dots of computable functions. Therefore, there is no Turing machine T generating all computable functions that C can generate by means of P .

²The theorems and proofs will be described more precisely in a separate paper.

Theorem 4 *Let C be a machine capable of processing an effective procedure P that exists according to Theorem 1. Then, there is no formal system containing all computable functions that C can generate by means of P .*

Proof. Let C be a machine capable of processing an effective procedure P that exists according to Theorem 1 and let S be any formal system with a predicate Q for computable (total) functions. Because a formal system can be regarded as a mechanical procedure or algorithm producing provable formulas (see Gödel 1965, p. 72), the formal system S produces an effective enumeration of provable formulas $Q(f_1), Q(f_2), \dots$ that contains all computable functions f_1, f_2, \dots in the formal system S . The machine C can generate a function g by applying the effective procedure P to the effective enumeration f_1, f_2, \dots of functions. According to Theorem 1 the function g is a computable function that is not contained in the enumeration f_1, f_2, \dots of functions. Therefore, there is no formal system S containing all computable functions that C can generate by means of P .

Because all programs of a programming language are finite sequences of a fixed finite number of symbols, they can be effectively enumerated, for example, in ascending length. This implies that there is an effective enumeration of all computable partial functions which need not be defined for all natural numbers. Thus, Theorem 1 implies that the function deciding whether or not a computable partial function is a total function is uncomputable, that is, not computable. If this function were computable, its application to an effective enumeration of all computable partial functions would yield an effective enumeration of all computable (total) functions. This contradicts Theorem 1. For these reasons, the function deciding whether a computable partial function is a total function is uncomputable. Thus, the function g in the proofs of Theorems 3 and 4, which is generated by the hypothetical machine C , is a value of this uncomputable function that is not contained in the given Turing machine T and the given formal system S , respectively. Therefore, Theorems 3 and 4 imply that the hypothetical machine C can compute values of this uncomputable function beyond the limits of any given Turing machine or formal system. This suggests the following definition for a new kind of machines.

Definition 1 A machine is called *creative* if it is capable of evaluating functions, that is, determining function values beyond the limits of any given Turing machine or formal system.

In view of Theorem 3 creative machines can process the informal concept of computable functions beyond the limits of any given Turing machine. In particular, the effective procedure P in the theorem can be regarded as a bridge to this informal concept, that is, to Turing's uncomputable "residue". The next section introduces hypotheses on the development of creative machines and their mode of operation. The hypotheses provide more information on the nature of Turing's "residue".

4 Hypotheses

Let P be a computer program whose properties are not known, that is, we have no or only partial knowledge about P . We can apply P to an input, for example, the natural number 1, which may produce the natural number 2. Thus, the execution of P produces knowledge which can be represented in function notation by

$$P(1) = 2. \quad (2)$$

We may apply the program P to another input, for example, the natural number 2, which may produce the natural number 3, that is, the knowledge

$$P(2) = 3. \quad (3)$$

From (2) and (3), we may assume by fallible inductive reasoning that

$$P(n) = n + 1 \quad (4)$$

holds for all natural numbers n . This simple example illustrates how knowledge about a program can be produced by the execution of the program and fallible inductive reasoning.

In order to develop a computer program a programmer usually applies all available knowledge. When the program is written, he tests it to verify whether it has the desired properties, that is, he has only partial knowledge about the program. In his tests he executes the program which produces further knowledge, for example, whether it generates an output from any input. Finally, he concludes on the basis of the tests and all available knowledge that the program has the desired properties. This conclusion can be regarded as fallible inductive reasoning. Thus, program development usually involves the application of all available knowledge, the execution of the program in tests

to produce further knowledge, and fallible inductive reasoning to conclude whether the program has the desired properties.

The expression $n + 1$ in (4) can be regarded as a program computing the natural number $n + 1$ from any natural number n . It can be constructed from the elementary knowledge that n and 1 are natural numbers and $x + y$ is a natural number for any natural numbers x and y . If we regard the variable n and the constant 1 in the expression $n + 1$ as nullary functions (without arguments) whose values are natural numbers, the expression $n + 1$ in (4) is just the composition of the functions n , 1, and $+$ that are contained in $n + 1$.

The composition of functions can be used as an elementary mechanism for the construction of sophisticated programs. For example, Ammon [1988] describes the automatic development of a program that proves theorems in mathematics whose complexity represented the state of the art in automated theorem proving. The program is also constructed on the basis of elementary functions that form the components of the program such as the “*left-side*” x of an equation $x = y$ (see Ammon 1988, p. 559, Table 2). Starting “from scratch” compositions of the elementary functions are used to construct “a sequence of more and more powerful partial methods [programs] each of which forms the basis for the construction of its successor until a complete method [program] is generated” [Ammon, 1988, p. 558, Abstract]. The elementary functions themselves which form the components of the program, for example, the “*left-side*” x of an equation $x = y$, can be constructed on the basis of the elementary instructions or functions of a programming language.

The preceding considerations formed a starting point for the following hypotheses about creative machines. The next sections provide further arguments. In the hypotheses, a programming language is meant to include elementary knowledge about the language, for example, the domains and ranges of its elementary instructions or functions which can be used to form compositions of the instructions or functions. Roughly speaking, the first hypothesis states that the knowledge in a creative machine is developed in a formally irreducible empirical process from a programming language.

Hypothesis 1 (Experience) A creative machine is the execution of a self-developing procedure including knowledge which starts from any universal programming language and any input. This process produces formally irreducible experience, that is, the self-developing procedure cannot be reduced to a formal system but the language and the input from which it starts. The

development of the procedure can be illustrated by the formula

$$L + P_t + E \rightarrow P_{t+1}, \quad (5)$$

where L is the universal programming language including knowledge about L itself, P_t is the procedure at time $t = 0, 1, 2, \dots$, P_0 is empty, and E stands for experience.

The second hypothesis refers to informal concepts such as the computable functions in Theorems 3 and 4.

Hypothesis 2 (Structure) The knowledge in a creative machine according to Hypothesis 1 contains informal concepts that are extensible beyond the limits of any given Turing machine or formal system.

The third hypothesis deals with inductive reasoning, that is, the construction of knowledge in a creative system.

Hypothesis 3 (Induction) In practice the construction of knowledge in a creative machine is achieved by informal inductive reasoning that is based on all available knowledge including informal concepts according to Hypothesis 2.

The last two hypotheses say that a creative system can revise all its knowledge and construct any knowledge.

Hypothesis 4 (Revision) A creative machine may revise all its knowledge but a universal programming language from which it starts according to Hypothesis 1. In principle, all knowledge is fallible and correctable by the machine.

Hypothesis 5 (Generality) A creative machine can in principle construct and verify any knowledge including all knowledge about itself as far as the knowledge can be constructed and verified.

5 Discussion

We discuss creative machines, in particular, in view of the Church-Turing thesis. The next section compares related work, for example, the Turing machine concept.

In order to apply an effective procedure P according to Theorem 1 in Section 2, a creative machine C according to Definition 1 in Section 3 needs an effective enumeration of computable functions, say E_1 . A simple example is an enumeration of functions f_1, f_2, \dots defined by $f_i(n) = i$ for all natural numbers i and natural numbers n . By applying the effective procedure P to E_1 the machine C can generate a computable function, say g_1 , which is not contained in E_1 according to Theorem 1. The function g_1 can be added to E_1 which yields an effective enumeration g_1, f_1, f_2, \dots , say E_2 . Thus, starting from any effective enumeration E_1 of computable functions a creative machine C can, by repeatedly applying P , generate a sequence E_1, E_2, \dots of effective enumerations of computable functions where E_n has the form $g_{n-1}, \dots, g_1, f_1, f_2, \dots$ for any natural number n greater than 1. In particular, each E_n contains another computable function g_{n-1} that is not contained in any preceding E_i , where i is a natural number less than n .

An effective enumeration E_1 of computable functions can be provided by a Turing machine, say T_1 generating the functions in E_1 or by a formal system, say F_1 , with a predicate for the computable functions in E_1 . By applying an effective procedure P according to Theorem 1 a creative machine C can generate a computable function, say g_1 . Theorems 1 and 2 in Section 2 imply that the function g_1 is neither generated by the Turing machine T_1 nor contained in the formal system F_1 . The incorporation of g_1 into the Turing machine T_1 and the formal system F_1 yields a more powerful Turing machine T_2 and a more powerful formal system F_2 . Thus, a creative machine C can, by repeatedly applying P , generate a sequence of more and more powerful Turing machines T_1, T_2, \dots and a sequence of more and more powerful formal systems F_1, F_2, \dots , respectively.

Up to any point in time a creative system C can only “know” a finite description of computable functions which can be represented by a Turing machine, say T_1 , generating the functions or in a formal system, say F_1 , with a predicate for the functions. As described above the creative machine C can, by repeatedly applying an effective procedure P according to Theorem 1 to T_1 or F_1 , generate more and more powerful Turing machines T_1, T_2, \dots and more and more powerful formal systems F_1, F_2, \dots , respectively. Thus, a creative system C can generate “knowledge”, that is, computable functions beyond the limits of its own knowledge in the Turing machine T_1 and the formal system F_1 .

According to our considerations preceding Definition 1 in Section 3, the propositional function (predicate), say d , deciding whether or not a com-

putable partial function of natural numbers is a total function is uncomputable. The preceding paragraphs in this section imply that a creative system can determine an unlimited number of values of the function d beyond its “knowledge”, that is, it can repeatedly generate computable functions beyond the limits of its “knowledge”, which is finitely describable, and the limits of a given Turing machine or formal system. The Church-Turing thesis states that every effectively calculable function is computable by a Turing machine (see Kleene 1952, pp. 317-323, 376-381). A creative machine can determine values of the function d beyond its finitely describable “knowledge” but it cannot calculate the value of the function $d(x)$ “for each value of x for which it is defined” (see Kleene 1987, p. 493), that is, it cannot determine for each computable partial function x whether x is a total function. Therefore, a creative system can compute values of uncomputable functions beyond the limits of given Turing machines or formal systems, but this has no bearing on what number-theoretic functions are effectively calculable (see Kleene 1987, p. 493). Furthermore, the computable functions generated by a creative machine cannot be specified *in advance* but must be generated by repeated applications of an effective procedure according to Theorem 1. This process involves the generation of a sequence of more and more powerful effective enumerations of computable functions, equivalent Turing machines, or equivalent formal systems each of which forms the basis for the generation of its successor. Roughly speaking, this process can be regarded as a self-developing procedure which must be generated step by step and cannot be reduced to a finite description given in advance (see Theorems 3 and 4 in Section 3). Kleene [1987, p. 493] requires that an “effective calculation procedure” or “algorithm” is “fixed in advance for all calculations” and that it is “possible to convey a complete finite description of the effective procedure or algorithm by a finite communication, in advance of performing computations in accordance with it”. For any computable (total) function that a creative system can generate it can give a complete finite description of its generation but there is no complete finite description for the generation of all computable functions it can generate (see Theorems 3 and 4).

Hypothesis 1 in Section 4 states that the knowledge in a creative machine is developed in a formally irreducible empirical process that starts from a programming language and any input. As described above, the development of knowledge is formally irreducible because it must be generated step by step and cannot be reduced to a finite description given in advance. For example, this process can involve the generation of a sequence of more and more

powerful Turing machines or formal systems each of which forms the basis for the generation of its successor (see Theorems 3 and 4). In a comparable process Ammon [1988] generates programs on the basis of elementary functions that form the components of the final programs. Starting from scratch, compositions of the elementary functions are used to construct a sequence of more and more powerful partial programs each of which forms the basis for the construction of its successor until a complete program is generated (see Section 4).

According to Hypothesis 2 a creative machine contains informal concepts such as computable functions. These concepts are processed by effective procedures such as an effective procedure according to Theorem 1. But according to Theorems 3 and 4 they are extensible beyond the limits of any given Turing machine or formal system. This means that informal concepts are processed by effective (finite) procedures although there is no complete formal (finite) description of these concepts.

According to Hypothesis 3 knowledge is constructed by informal inductive reasoning that is based on all available knowledge. As described in Section 4 a model of such reasoning processes is the development of computer programs which are empirically verified in tests and then used in practice. Such an empirical verification requires only limited resources such as time. This means that informal inductive reasoning is regarded as more efficient in practice than sophisticated formal procedures. Because all available knowledge can be used there is no sophisticated formal method for the construction of knowledge that can specified in advance.

The fourth hypothesis implies that a creative machine may start from scratch, that is, from any programming language and any input. Thus, it can revise any knowledge that proves to be false in a specific case.

The last hypothesis says that a creative machine can construct and verify any knowledge including knowledge about itself. Here, “any knowledge” includes any knowledge that can be constructed by a human and is finitely describable.

According to Definition 1 in Section 3 a creative machine is capable of evaluating functions beyond the limits of any given Turing machine or formal system. This means that it is capable of evaluating uncomputable functions. Hypotheses 3 and 4 imply that this evaluation ordinarily involves empirical inductive reasoning that is fallible and correctable, that is, the values $f(x)$ of uncomputable functions f a creative machine assigns to arguments x may be false but can be corrected by the machine. An example of an uncom-

putable function is the uncomputable propositional function (predicate) that determines whether a computable partial function is a total function. The development of computer programs provides a model for the evaluation of this uncomputable predicate because, for example, a program must produce an output for any input for which the program is defined.

How can we use our insight into the nature of Turing's residue, in particular, into the evaluation of uncomputable functions to implement a creative machine? Turing [1969] writes: "Our task is to discover the nature of this residue as it occurs in man, and to try and copy it in machines." According to our hypotheses we must choose a programming language and implement knowledge about this language, for example, the domains and ranges of the elementary functions or instructions it contains. In order to reduce the complexity of a first implementation we should reduce the number and complexity of the informal concepts to be implemented. A concrete project might aim to prove theorems in a mathematics textbook on the basis of preceding theorems and proofs (see Ammon 1988). According to our approach no formal system, which eliminates Turing's residue, should be used but the ordinary representation of theorems and proofs should be modeled. Another possible field of application might aim at an automatic development of programs. An implementation of a creative machine would be general and complete if it can change all its source code and develop new knowledge from scratch, that is, from a universal programming language. Obviously, Ammon [1988] is far from a general and complete system in our sense.

The theorems and hypotheses on creative machines have epistemological implications. For example, informal concepts such as computable functions are extensible beyond any formal limits. Because a creative machine can construct and revise any knowledge starting from any programming language and any input there is only a rather limited general description of its structure and development, in particular, its starting point. This is comparable with Piaget's [1970, p. 704] view: "Knowledge, then, at its origin, neither arises from objects nor from the subject, but from interactions - at first inextricable - between the subject and those objects."

6 Related Work

Ordinarily, the word "machine" refers to a Turing machine which is a formalization of the concept of an *algorithm* or effective *procedure* (see Hopcroft

and Ullman 1979, pp. 146-147). In contrast, Theorems 3 and 4 contain no assumptions on the machine C but its capability to process an effective procedure P according to Theorem 1. In particular, the effective procedure P and the machine C are regarded as different entities. Thus, Theorems 3 and 4 can be applied to any machine capable of processing the effective procedure P .

If one requires that a creative machine can execute any Turing machine it can compute any computable function and evaluate uncomputable functions as described in this paper.

One might ask whether the machine C in Theorem 3 can be modeled by a Turing machine generating computable functions, say T , into which an effective procedure P according to Theorem 1 is incorporated. But according to Theorem 3 the machine C could apply P to T to generate a computable function that is not generated by T . Therefore, the Turing machine T cannot model the machine C , that is, it cannot generate a computable function beyond the limits of any given Turing machine.

Turing [1936, p. 232] writes: “the motion of the machine ... is *completely* determined by the configuration” which comprises a condition from a “finite number of conditions” and a “scanned symbol”. The behavior of a creative machine cannot be completely determined in advance because it contains informal concepts such as computable functions which cannot be reduced to a finite description given in advance (see Theorems 3 and 4). But at any moment and at the most basic level the next “instruction” is completely determined by the present state of a creative machine, which can be regarded as a finite string of symbols or binary digits, and the input processed by the creative machine.

Turing [1936, pp. 249-250] attempts to show that his machines can compute “all numbers which would naturally be regarded as computable” and supposes that the “number of states of mind” is finite. Gödel [1990b, p. 306] points out that “mental procedures” may “go beyond mechanical procedures ... mind, in its use, is not static, but constantly developing, i.e., that we understand abstract terms more and more precisely as we go on using them ... although at each stage the number and precision of the abstract terms at our disposal may be finite, both (and, therefore, also Turing’s number of distinguishable states of mind) may converge toward infinity in the course of the application of the [mental] procedure”. The “abstract terms” refer to abstract concepts such as “a certain concept of computable function” (Gödel 1990a, p. 271) and “the use of abstract terms on the basis of their meaning”

(Gödel 1965, p. 72, footnote **). Creative machines can process informal concepts such as computable functions. Beyond any given Turing machine or formal system they can generate an unbounded number of such functions which can be regarded as a part of the extensible meaning of this informal concept.

Our paper is no contribution to formal logic. Rather, we leave formal logic on the basis of Theorems 3 and 4 and our hypotheses on creative machines. We allow creative machines to apply Cantor’s diagonalization to informal concepts such as computable functions in Turing machines and formal systems which are restricted to enumerable subsets of nonenumerable sets that are contained in these concepts. At any point in time a creative machine can only contain such an enumerable subset but it can generate an unbounded number of more and more extended subsets. A creative machine can simply refute the claim that it is modeled by a Turing machine or a formal system by applying an effective procedure according to Theorem 1 to the machine or system.

We argue that any “intelligent” machine should be capable of processing informal concepts such as computable functions beyond the limits of any given Turing machine or formal system, that is, it should be creative. Formal descriptions restrict the generality of a machine, in particular, informal concepts such as computable functions.

Gödel’s theorem says that every sufficiently powerful consistent formal number theory contains an undecidable proposition. Lucas [1961] argues that mind cannot be modeled by a Turing machine because he *knows* that the undecidable proposition is true. Putnam points out that Lucas cannot prove the prerequisite of consistency in Gödel’s theorem (see Shapiro 1998, pp. 282-284). The machine C in Theorem 3 can process an effective procedure P that produces a computable function g from an effective enumeration of such functions. This enumeration is comparable with the consistent number theory in Gödel’s theorem and the function g with the undecidable proposition. If a creative machine “knows” an effective enumeration of computable functions it can also “know” the computable function g which is not contained in the enumeration. The prerequisite of an effective enumeration of computable functions can be satisfied because there are simple examples of such enumerations and, up to any point in time, a creative machine C can only “know” a finite description of computable functions which can be represented by such an enumeration (see Section 5). Without any assumptions on the structure of mind but its capability to process an effective proce-

ture according to Theorem 1 these considerations also apply to humans who can thus extend their knowledge beyond any formal limits. Post’s [1944, p. 295] conclusion from Gödel’s theorem is that “*mathematical thinking is ... essentially creative*” (see Shapiro 1998, pp. 291-292, “creative step”).

7 Conclusion

Turing points out that intelligence requires a “residue” called “initiative” that is not captured by his machines. We introduced the concept of a creative machine by requiring that it can evaluate uncomputable functions in a certain sense. It is capable of extending informal concepts such as (total) computable functions beyond any given formal description which is restricted to incomplete enumerable parts of these concepts. We argue that “intelligent” machines should be capable of processing such concepts. Hypotheses on creative machines say that Turing’s uncomputable “residue” is the execution of a self-developing procedure that starts from any programming language and any input. This process, which produces formally irreducible experience, can be regarded as a new use of computers.

Acknowledgments. The author wishes to thank colleagues for valuable comments.

References

- [Ammon, 1988] Kurt Ammon. The automatic acquisition of proof methods. In *National Conference on Artificial Intelligence, St. Paul*, San Mateo, Calif., 1988. Morgan Kaufmann.
- [Gödel, 1965] K. Gödel. On undecidable propositions of formal mathematical systems - POSTSCRIPTUM. In M. Davis, editor, *The Undecidable*. Raven Press, New York, 1965.
- [Gödel, 1990a] K. Gödel. Gödel 1972: On an extension of finitary mathematics which has not yet been used. In S. Feferman et al., editors, *Collected Works: Publications 1938-1974*, volume 2. Oxford University Press, New York, 1990.

- [Gödel, 1990b] K. Gödel. Gödel 1972a: Some remarks on the undecidability results. In S. Feferman et al., editors, *Collected Works: Publications 1938-1974*, volume 2. Oxford University Press, New York, 1990.
- [Hilbert and Ackermann, 1928] D. Hilbert and W. Ackermann. *Grundzüge der theoretischen Logik*. Springer, Berlin, 1928.
- [Hopcroft and Ullman, 1979] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.
- [Kleene, 1952] S. C. Kleene. *Introduction to Metamathematics*. North-Holland, Amsterdam, 1952.
- [Kleene, 1987] S. C. Kleene. Reflections on Church’s thesis. *Notre Dame Journal of Formal Logic*, 28(4):490–498, 1987.
- [Lucas, 1961] J. R. Lucas. Minds, machines, and Gödel. *Philosophy*, 36:112–137, 1961.
- [Piaget, 1970] J. Piaget. Piaget’s theory. In P. H. Mussen, editor, *Carmichael’s Manual of Child Psychology*. John Wiley, New York, 1970.
- [Post, 1944] E. Post. Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society*, 50(5):284–316, 1944.
- [Shapiro, 1998] S. Shapiro. Incompleteness, mechanism, and optimism. *The Bulletin of Symbolic Logic*, 4(3):273–302, 1998.
- [Sieg, 1994] W. Sieg. Mechanical procedures and mathematical experience. In A. George, editor, *Mathematics and Mind*. Oxford University Press, New York, 1994.
- [Turing, 1936] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. In *Proceedings of the London Mathematical Society*, volume 42 of *series 2*, pages 230–265, 1936.
- [Turing, 1969] A. M. Turing. Intelligent machinery. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*. Edinburgh University Press, 1969.

[Turing, 1986] A. M. Turing. Lecture to the London Mathematical Society on 20th February 1947. In B. E. Carpenter and R. W. Doran, editors, *A. M. Turing's ACE report of 1946 and other papers*. MIT Press, Cambridge, 1986.